Technical Disclosure Commons

Defensive Publications Series

June 27, 2019

SYSTEM AND METHOD FOR EFFICIENT PROCESSING OF OXPd HIDREPORT EVENTS IN A MASSIVE CLOUD-SCALE DEPLOYMENT

HP INC

System and Method for Efficient Processing of OXPd HIDReportEvents in a Massive Cloud-Scale Deployment

Abstract

The system and method disclosed in this document enables efficient processing of OXPd Accessories Service HIDReportEvents in a massive scale cloud deployment. Key components of the system include:

- Utilization of a traditional load-balancing scheme to provide the single "front-end" callback endpoint for receiving all OXPd HIDReportEvents from all devices. (E.g. the AWS API Gateway Service could be used.)
- Multiple instances of "OXPd Accessories Callback Handlers" whose instances can increase/decrease depending on the current system load. (E.g. AWS Lambda Service could be used.)
- A database/repository for managing individual and uniquely identifiable HID Reports. (E.g. the AWS DynamoDB service could be used.)
- A "Report Event Assembler" component to perform the business logic of processing the individual HID Reports from the repository and assembling complete card IDs. (E.g. the AWS Lambda service could be used.)
- A Message Queue or notification service for managing and routing assembled card IDs. (E.g. the AWS SMS Service could be used.)
- A "Card ID Processor" to perform the business logic that depends on complete card IDs. (E.g. the AWS Lambda service could be used.)

System and Method Description

The Open Extensibility Protocol (Device) (OXPd) is a protocol that serves as the basis for one of the extensibility platforms that printers and MFDs implement to enable 3rd party solutions to use and interact with device functionality. The OXPd protocol consists of a set of web-services that are implemented on the device, and these web-services are exposed to solution partners that have access to the OXPd SDK.

One of the OXPd services provides access to HID-class USB accessories that may be attached to a printer/MFD. This service is the OXPd Accessories Service, and its interface allows a 3rd party solution to manage, write-to, and read-from the HID-class USB accessory.

The OXPd Accessories Service protocol defines a specific pattern for how the device delivers individual HID InputReports to the appropriate 3rd party solution. This delivery is accomplished by the device making an HIDReportEvent callback to an endpoint registered by the 3rd party solution, and this event callback may contain from 1-n individual HID InputReports.

A common use case for solutions that utilize the OXPd Accessories service is to detect RF/proximity card reads at the device and use the information read from the proximity card to identify an individual user. The data read from the proximity card is typically reported in a sequence of small

individual buffers, and therefore it requires processing/assembly of the individual buffers in the backend before the data can be utilized.

In a small-scale/enterprise deployment, it is relatively straightforward to implement a solution for handling the card reads that uses a single handler to receive all OXPd HIDReportEvents from all devices and then assemble the individual HID reports into corresponding proximity card IDs.

However, in a massive scale cloud deployment scenario it is unlikely that a solution deployment can depend on a single handler to process all OXPd HIDReportEvent callbacks because such a solution does not work with the traditional load-balancing schemes that enable the horizontal scaling of cloud deployments.

A massive cloud-scale deployment will have many 100s to 1000s of individual printers/MFDs provisioned to participate in the solution. Each of these devices will be making OXPd Accessories Service HIDReportEvent callbacks to the registered solution endpoint. These HIDReportEvents are all uniquely traceable to the device that generated them (enabled through the use of the OXPd Accessories Service “ServerContextId” property) and will each contain one or more individual HID Read reports.

The load-balancing front end receives all HIDReportEvents, which are in turn forwarded to arbitrary “OXPd Accessories Callback Handlers” for further processing.

When an OXPd Accessories Callback Handler instance receives a HIDReportEvent, it performs a simple unpacking of the event into its 1-n individual HID Report buffers. Each of the individual HID reports is inserted into the database. Metadata for each raw report buffer entry includes:

- Which device the report is from (via standard OXPd Accessories Service HIDReportEvent “ServerContextId” property)
- The temporal position of the report in the overall sequence of reports (via the standard OXPd Accessories Service HIDReportEvent “Ordinal” property plus report-index)

As entries are inserted into the database, the “Report Event Assembler” component is notified. Its responsibility is to process the individual raw report buffers into whatever the contiguous card data format is, typically a card ID number. Such assembly is possible by using the metadata of each individual entry. When a contiguous card data payload is available, the “Report Event Assembler” will push a “CardReadComplete” message into the queue. This event will identify from which device the read is from and contain the assembled card data.

The appearance of a CardReadComplete message will trigger the “Card ID Processor” component to activate and ultimately perform the necessary solution business logic that should occur when a card ID has been read.

The diagram shown below in Figure 1 provides a visual description of the system and method described above.

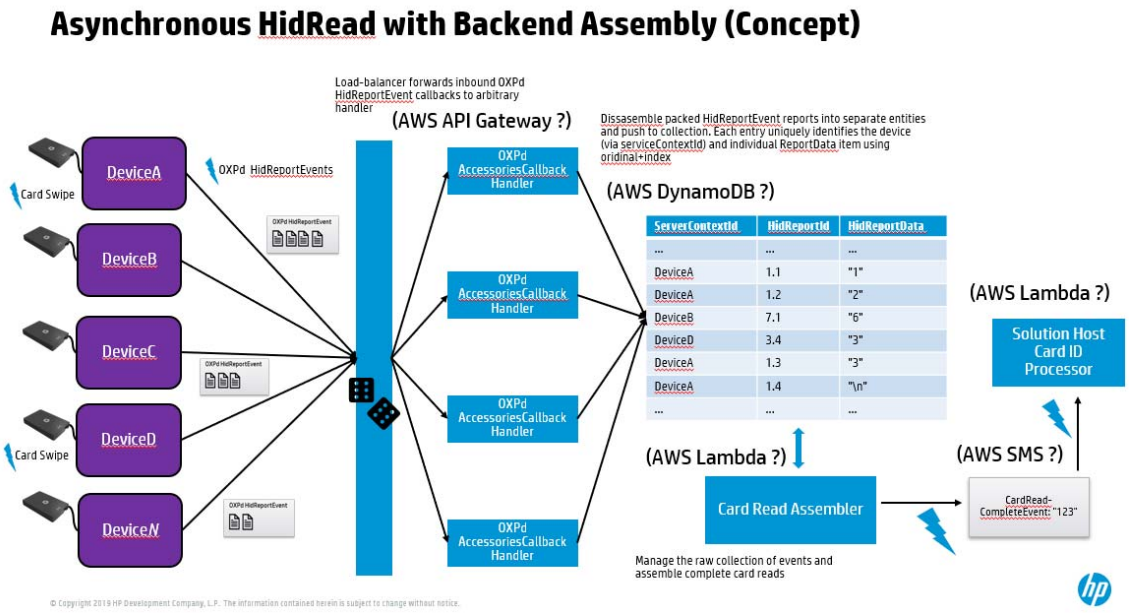

Figure 1

The primary advantage of this solution is in cost savings and scalability of the processing required to support massive cloud-scale deployment of OXPd solutions that need to use the OXPd Accessories Service for handling HIDReportEvents.

Without using the mechanism disclosed here, monolithic instances that process the whole "sequence" must be deployed. This complicates the deployment of the solution overall since simple load-balancing algorithms and scalability services of popular cloud platforms cannot be used in an effective manner.

Disclosed by Travis M Cossel, HP Inc.